

# ПРОТОТИП РАСПРЕДЕЛЕННОЙ ПРОГРАММНОЙ ТРАНЗАКЦИОННОЙ ПАМЯТИ DSTM\_P1

*И.Г. Данилов*

*Технологический институт Южного федерального университета, Таганрог*

## **Введение**

В последние годы в связи с развитием вычислительных архитектур и межсистемных соединений все больший интерес получают исследования в области совершенствования существующих моделей и алгоритмов параллельного программирования с учетом новых технических возможностей. В первую очередь это связано со стремлением добиться более эффективного использования имеющихся вычислительных ресурсов. В настоящее время существуют актуальные в плане исследования и практического применения технологии для эмуляции общей памяти на распределенных системах. Одной из таких технологий является транзакционная память. Транзакционная память (*Transactional Memory*, ТМ) – это альтернативный традиционным методам синхронизации механизм, позволяющий частям программы выполняться в изоляции, независимо от других параллельно выполняемых задач [1]. Отказ от блокировок и использование механизмов транзакционной памяти при написании многопоточных приложений имеет следующие преимущества:

- 1) облегчение процесса разработки многопоточного программного обеспечения;
- 2) прозрачное для пользователя решение трудноуловимых проблем блокировок (взаимоблокировки, «живые» блокировки, инверсия приоритетов и т. п.);
- 3) в перспективе – повышение производительности и эффективности использования вычислительных ресурсов (за счет изолированности транзакций);
- 4) возможность простой композиции программного кода.

Большинство исследований [2] в данной области рассматривает не распределенные, а кэш-когерентные системы или системы с общей памятью, что уменьшает возможность масштабирования данного подхода. Лишь совсем недавно стал рассматриваться вопрос о расширении границ использования транзакционной памяти для ее применения в широко масштабируемых распределенных многомашинных конфигурациях. Решение этого вопроса позволит более просто и эффективно использовать доступные кластерные вычислительные ресурсы, применяя устоявшиеся технологии создания многопоточного ПО.

В статье [3] была высказана идея по использованию механизмов транзакционной памяти в системах, построенных на основе облачных технологий. Создан консорциум [4], поддерживаемый Европейским сообществом, для решения различных вопросов, связанных с данной тематикой.

На основе описанных выше идей автор предлагает создать облачную платформу (PaaS), по запросу предоставляющую в качестве сервиса ресурсы кластерной системы для многопоточных приложений пользователя. Первым шагом к реализации данной платформы, по мнению автора, является создание прототипа распределенной программной транзакционной памяти (*Distributed Software Transactional Memory*, DSTM), основные тонкости реализации которого описываются в следующих разделах.

## Распределенная программная транзакционная память

Многие современные системы DSTM используют модель потока данных, в которой сами транзакции неподвижны, а объекты данных перемещаются по сети. Как правило, такие системы включают в себя три основных элемента. Первый элемент – распределенный алгоритм версионирования данных, который должен предоставлять гарантии атомарности для всех операций над данными в рамках транзакции и гарантировать консистентность (непротиворечивость) состояния общих данных для всех транзакций. Второй – распределенный протокол когерентности кэшэй (*distributed cache-coherence* протокол, СС-протокол). Когда транзакция пытается получить доступ к объекту по сети, распределенный СС-протокол должен определить местонахождение последней актуальной версии объекта и переместить его копию к запрашивающей транзакции. Третий элемент – стратегия разрешения конфликта. Существующие реализации DSTM откатывают одну из конфликтных транзакций, исходя из определенных критериев.

### Реализация прототипа

При проектировании прототипа DSTM\_P1 были выдвинуты следующие основные требования: (1) на распределенной кластерной системе прототип должен эмулировать единое адресное пространство для потоков приложения; (2) пользователь имеет возможность загружать (без изменения) в систему существующие исходные коды параллельной программы для ее выполнения; (3) в функциях потока приложения производятся только вычисления и обращения к разделяемой памяти. Опишем вспомогательные инструменты, используемые при реализации прототипа:

1. *Библиотека сетевых коммуникаций GASNet*. Самым узким местом, влияющим на производительность системы распределенной транзакционной памяти в целом, является протокол когерентности кэшэй. Для уменьшения сетевых накладных расходов существует необходимость в разработке специфических для многомашинных конфигураций широко масштабируемых алгоритмов. Ряд подходов был рассмотрен в [2]. Вместе с этим следует отметить, что к настоящему времени большинство современных вычислительных кластерных платформ оснащается специализированным высокоскоростным межсистемным интерконнектом (например, высокоскоростная коммутируемая последовательная шина Infiniband). Данные технологии предоставляют возможность низколатентного сетевого удаленного прямого доступа к памяти без задействования центрального вычислительного устройства (*Remote Direct Memory Access, RDMA*), и их использование для эффективной реализации системы DSTM является критически важным. В качестве коммуникационного сетевого уровня в описываемом прототипе используется библиотека односторонних коммуникаций GASNet [5]. Данная библиотека разрабатывалась для реализации современных PGAS-языков и предоставляет высокопроизводительные независимые от нижележащего сетевого оборудования коммуникационные примитивы, в частности примитивы для RDMA-операций. GASNet-программа представляет собой обычную SPMD-программу. Память удаленного вычислительного узла адресуется с помощью номера этого узла (нумерация ведется целыми числами, начиная с нуля) и адреса начала зарезервированного для удаленного доступа сегмента виртуальной памяти этого узла. Все алгоритмы работы с памятью (резервирование, сегментация, доступ) прозрачны для пользователя и эффективно реализуются в контексте используемого сетевого оборудования.

2. *Библиотека языковых трансформаций LLVM*. Реализация возможности пользовательского доступа к распределенной памяти, так же как и к транзакционной, требует от разработчика либо внесения изменений в ядро ОС, либо внесения изменений в конструкции языка, либо реализации специальных пользовательских библиотек. Существует также и другой способ: каким-либо образом «перехватывать» вызовы приложе-

ния по обращению к памяти. Данный способ (также как и изменение ядра ОС) позволяет не изменять пользователю уже написанный код приложения, что является одним из требований к описываемой системе. Поэтому этот прием используется в DSTM\_P1 и реализуется с помощью LLVM [6]. LLVM – универсальная инфраструктура для анализа, оптимизации и трансформации программ. В ее основе лежит промежуточное представление кода (*Intermediate Representation, IR*) в SSA форме, над которым можно производить различные трансформации во время компиляции, компоновки и выполнения программы. Результирующий LLVM-биткод (IR в байтовой форме) может быть оттранслирован в исполняемый машинный код как статически, так и динамически (при использовании JIT-компиляции). Работа с памятью в LLVM осуществляется с помощью двух инструкций: `load` (загрузка значения из памяти по указателю) и `store` (сохранение в памяти значения по указателю). Такая архитектура позволяет заменять вызовы по обращению к памяти на уровне LLVM IR в вызовы соответствующих библиотек.

Рассмотрим более подробно решенные с помощью данных инструментов основные вопросы по реализации прототипа DSTM\_P1.

1. *Поддержка транзакционной памяти на уровне языка.* Реализация любого алгоритма транзакционной памяти в первую очередь требует введения в используемый язык программирования средств поддержки атомарных конструкций и управления доступом к памяти. В языках с управляемой средой выполнения, таких как Java, используются техника аннотации атомарных объектов и методов, генерация, инструментирование и исполнение байт-кода класса во время выполнения программы. Примеры реализации систем STM на Java можно найти в [7, 8]. При использовании обычных языков программирования, таких как C и C++, предполагается применение либо специального API библиотеки транзакционной памяти [9], либо макросов языка совместно с динамическим бинарным инструментированием кода [10]. Кроме того, возможно внесение изменений в один из существующих компиляторов языка с целью поддержки атомарных конструкций в пользовательских программах. Последний подход используется в Dresden TM Compiler, DTMC [11] – это измененная версия компилятора GCC на базе LLVM. DTMC используется совместно с DSTM\_P1 следующим образом: с помощью него компилируется загружаемое пользователем в виде исходных кодов приложение, написанное с применением атомарных конструкций `__tm_atomic{}` на языке C или C++, в язык промежуточного представления LLVM IR; далее полученный биткод трансформируется с помощью утилиты Tanger [12] в биткод, который содержит соответствующие вызовы функций библиотеки транзакционной памяти для всех инструкций `load/store` атомарного блока.

2. *Реализация потоковой библиотеки.* В настоящее время прототип поддерживает программный код, написанный с использованием API широко распространенной и хорошо стандартизированной библиотеки Pthreads. Реализованы в виде биткод-библиотеки `dstm_pthread` обертки основных функций: создания потока (`pthread_create`) и ожидания завершения потока (`pthread_join`). Пользователь загружает в систему исходные коды параллельной программы, которая компилируется с помощью DTMC. Результирующий LLVM-биткод обрабатывается Tanger и компонуется с библиотекой `dstm_pthread`. Далее функция `main` на лету компилируется в машинный код на главном узле (GASNet-узел с нулевым номером) и выполняется. При вызове `pthread_create` определяется код функции потока, который распределяется в системе с помощью модуля балансировки нагрузки и выполняется в виде задания на менее загруженном узле. Предполагается, что функция потока написана с использованием атомарных конструкций `__tm_atomic{}` в местах обращения к разделяемой памяти. Однако это приводит к необходимости изменения кода пользовательского приложения. Данную проблему можно решить путем ав-

томатической замены конструкций критических регионов (`pthread_mutex_lock/unlock`) на атомарные конструкции.

3. *Виртуальное адресное пространство.* Вопрос организации распределенной разделяемой памяти был очень хорошо изучен в последнюю декаду ушедшего столетия. Были предложены различные реализации: (1) на основе страниц памяти [13]; (2) на основе специальных разделяемых переменных, что требует языковой поддержки; (3) на основе объектно-ориентированных технологий. При любом используемом подходе необходимо решить задачу эмуляции единого адресного пространства, т.е. ввести определенную логическую адресацию памяти. В DSTM\_P1 эксплуатируется свойство неполноты текущей реализации виртуальной адресации для 64-битных платформ: современная аппаратная реализация трансляции адресов для 64-битных платформ использует только первые 48-бит виртуального адреса, остальные остаются незадействованными [14]. Эти незадействованные биты (с 48 по 63) применяются для хранения номера GASNet-узла, к которому этот адрес относится; остальные биты – обычный виртуальный адрес, лежащий в границах зарегистрированного на этом узле GASNet-сегмента.

4. *Доступ к памяти.* При использовании описанной выше адресации необходимо было решить два основных вопроса: (1) каким образом разыменовывать логический адрес в виртуальный адрес ОС, без изменения пользовательского приложения и введения дополнительного API; (2) каким образом выделять память по запросу приложения. В описываемом прототипе первый вопрос решается с помощью LLVM-трансформаций: для участков с атомарным блоком кода (конструкция `__tm_atomic{}`) соответствующие инструкции `load/store` с помощью Tanger ретранслируются в вызовы библиотеки транзакционной памяти; для участков с нетранзакционным кодом инструкции `load/store` ретранслируются в вызовы соответствующих функций одного из модулей прототипа, в которых проводится проверка старших битов адреса (для выяснения номера узла, в сегменте которого этот адрес расположен) с последующим обнулением этих битов и доступом к памяти с помощью RDMA-операций библиотеки GASNet. Для решения второго вопроса, как и в случае с библиотекой Pthread, были реализованы обертки к библиотечным функциям для работы с памятью в языке C: `malloc` и `free`. Полученная LLVM-библиотека `dstm_malloc` компонуется с биткодом пользовательского приложения до его выполнения. В настоящий момент реализация функции `malloc` выделяет память из доступной памяти зарегистрированного библиотекой GASNet сегмента вызывающего узла и возвращает указатель в формате описанного выше логического адреса. В дальнейшем планируется улучшить алгоритм так, чтобы подсистема памяти была полностью распределенной.

5. *Реализация библиотеки распределенной программной транзакционной памяти.* В настоящее время ведется работа по созданию основной части прототипа: библиотеки распределенной программной транзакционной памяти. Библиотека `dstm_lib_p1` реализует унифицированный интерфейс, который предоставляется утилитой трансформации Tanger, и включает в себя три основных компонента: 1) *ядро* - компонент реализующий основную логику распределенного алгоритма версионирования данных; 2) *протокол*, реализующий распределенный протокол когерентности кэшей; 3) *менеджер конфликтов*, реализующий логику системы при возникновении конфликтов между транзакциями. В качестве основного алгоритма ядра системы был выбран хорошо описанный, масштабируемый и эффективный алгоритм D-TL2 [8]. В качестве распределенного протокола когерентности кэшей и менеджера конфликтов была выбрана комбинация протокола DHTC и «жадного» менеджера конфликтов, описанная в работе [15]. DHTC – эффективный протокол, основанный на технологии распределенных хэш-таблиц и оверлейных сетей. Этот выбор обусловлен тем, что пиринговые протоколы обладают

необходимыми для любой облачной системы свойствами масштабируемости, отказоустойчивости и гибкости (в плане включения/исключения узлов в/из системы, поиска объектов данных).

### **Заключение**

Актуальность настоящей работы состоит в обозначении подходов к построению программной системы с эмуляцией единого образа на многомашинных платформах и возможностью предоставления ресурсов данной системы для многопоточных приложений пользователей в виде сервиса.

### **Литература**

1. Larus J., Kozyrakis C. Transactional Memory // In: Communications of the ACM. 2008. Vol. 51 (7). P. 80-88.
2. Bocchino R. L., Adve V. S., Chamberlain B. L. Software transactional memory for large scale clusters // In Proc. ACM SIGPLAN Symp. on Principles and Practice of Parallel Prog (PPOPP 2008), Salt Lake City, UT, USA. 2008. P. 247-258.
3. Romano P., Rodrigues L., Carvalho N., Cachopo J. Cloud-TM: harnessing the cloud with distributed transactional memories // In: ACM SIGOPS Oper. Syst. Rev. 2010. Vol. 44 (2). P. 1-6.
4. A Novel Programming Paradigm for the Cloud. URL – [<http://www.cloudtm.eu/home>].
5. Bonachea D. GASNet Specification, v1.1 // URL – [<http://techreports.lib.berkeley.edu/accessPages/CSD-02-1207>].
6. The LLVM Compiler Infrastructure. URL: <http://llvm.org/>.
7. Herlihy M., Luchangco V., Moir M. A flexible framework for implementing software transactional memory // In Proc. of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (OOPSLA '06), ACM, New York, NY, USA. 2006. P. 253-262.
8. Saad M.M., Ravindran B. Hyflow: A high performance distributed software transactional memory framework // In Proc. of the 20th IEEE International Symposium on High Performance Distributed Computing (HPDC '11), San Jose, California, USA. 2011. P. 265-266.
9. Saha B., Adl-Tabatabai A.-R., Hudson R.L., Minh C., Hertzberg B. McRT-STM: a high performance software transactional memory system for a multi-core runtime // In Proc. of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP '06), ACM, New York, NY, USA. 2006. P. 187-197.
10. Olszewski M., Cutler J., Steffan J. G. JudoSTM: A Dynamic Binary-Rewriting Approach to Software Transactional Memory // In Proc. of the 16th International Conference on Parallel Architecture and Compilation Techniques (PACT '07), IEEE Computer Society, Washington, DC, USA. 2007. P. 365-375.
11. Dresden TM Compiler. URL: <http://tm.inf.tu-dresden.de/>.
12. Felber P., Fetzer C., Süßkraut M., Müller U., Sturzrehm H. Transactifying Applications using an Open Compiler Framework // In Proc. of the 2nd ACM SIGPLAN Workshop on Transactional Computing (TRANSACT '07), Portland, Oregon. 2007.
13. Amza C., Cox A. L., Dwarkadas S., Keleher P., Lu H., Rajamony R., Yu W., Zwaenepoel W. TreadMarks: Shared memory computing on networks of workstations // In: IEEE Computer. 1996. Vol. 29 (1). P. 18-28.
14. Canonical form addresses. URL – [[http://en.wikipedia.org/wiki/X86-64#Canonical\\_form\\_addresses](http://en.wikipedia.org/wiki/X86-64#Canonical_form_addresses)].
15. Zhang B., Ravindran B. Location-Aware Cache-Coherence Protocols for Distributed Transactional Contention Management in Metric-Space Networks // In Proc. of the 28th

IEEE International Symposium on Reliable Distributed Systems (SRDS '09), IEEE Computer Society, Washington, DC, USA. 2009. P. 268-277.